Journal of Engineering and Applied Sciences 12 (Special Issue 5): 7057-7062, 2017
ISSN: 1816-949X

Program Statement Parser for Computational Programming Feedback

[1, 2]S. Suhailan, [1]S. Abdul Samad, [1]M.A. Burhanuddin and [2]A.H. Nazirah
[1]Faculty of ICT, Universiti Teknikal Malaysia Melaka, 76100 Melaka, Malaysia
[2]Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin,
22200 Terengganu, Malaysia

Abstract: Programming skills can be trained by doing lots of computational programming exercises. Such practice does not only strengthen the understanding of a programming language syntax, but also developing the skills of problem solving using computer's logics. Unfortunately, automated feedback in rectifying problem solving difficulty is costly to be built due to the diversities of question's requirement. Among popular methods to support such feedback are through dynamic testing, solution template and intelligent agents. However, these approaches require additional resources to be prepared in advance of training session. Thus a simple and immediate method to associate feedback with current programming answering difficulty is addressed in this research. It enables each line of computer statements that having semantic mistake to be instantly associated with live expert's feedback using program-statement parser. Experiment was done using 793 solution attempts of in answering a computational programming question. The results show that feedback can be quickly provided on specific program's statement. Although the expert's feedback was only provided on 1% of the overall programs' attempts, the same feedback was successfully replicated up to 33% of the total programs' attempts that contain similar mistakes. This approach can be considered efficient as it does not require feedback's resources to be prepared in advance on each computational programming question. Furthermore, it provides mechanism to support expert's live feedback that can be provided during a lab session into a reusable automated feedback.

Key words: Automated computer assessment, programming feedback, Semantic error, text-based parser, Malaysia

INTRODUCTION

The assessment of a computational programming solution is commonly done by using dynamic testing approaches (Helminen *et al*., 2013; Kurnia *at al*., 2001; Tang *et al*., 2010; Papancea *et al*., 2013; Denny *et al*., 2011). Test cases are used to execute a program using a certain input with an expected output. Based on mismatches of the test output, a feedback can be produced to the user to highlight the mistake. However, as the approach does not evaluate the program structure, the feedback may not be sufficient to highlight the errors. On the other hand, semantic parser is an analysis of the code structure based on its meaning or logics of specific solution goals or requirements. In order to represent a requirement of specific problem-solving exercise, solution template is commonly used to validate the correctness of a computer program. The solution template is a sample of the computer program that solves the problem-solving exercise. Such template will represent basic structure of computer program semantic solution that will be used to match student's computer program in providing specific feedback. The earlier version of solution template approach uses syntax tree parser to find mismatches between a computer program and a template such as PROUST (Johnson, 1990), GOES (Sykes and Franek, 2003), JITS (Suarez and Sison, 2008), MEDD (Gulwani *at al*., 2014) and EML (Vaessen *et al*., 2014). The discrepancy between them can be reported as a feedback to guide which part of the computer program need to be revised.

Many of the researchers that use pre-defined solution template strategies claim that the highest success rate in helping student solving programming problems are achieved. Although, such approach guide students towards to the correct answer, this kind of feedback may not be effective in the context of learning outcome. This is due to the nature of template matching that may be abused by using trial and error strategy to get closer to the solution and finish the exercises without really understand the reason why the error has occurred (Tam, 2011). Furthermore, syntax-based parser tends to be costly to be built especially in satisfying varieties of computational problem's specification (Naser, 2008). Usually, more than single solution template is required on a single question especially in supporting the diversity of correct solution logics.

Other approach for computational programming feedback is by using intelligent agent as a tutoring system

Corresponding Author: S. Suhailan, Faculty of ICT, Universiti Teknikal Malaysia Melaka, 76100 Melaka, Malaysia

(Berdu *et al*., 2008; Mungunsukh and Cheng, 2002; Noranis and Azuan, 2013; Rafique *et al*., 2011; Mungunsukh and Cheng, 2002; Song *et al*., 1997). However, this approach requires a controlled environment that must be well defined in advanced. If a new environment such as new computational programming question is to be added, the agent needs to be trained with additional efforts. Thus, it is not suitable to have a dynamic feedback that can be tailored to a dynamic set of computational programming question that will kept changing from time to time.

In order to decrease workload in preparing pre-designed feedback on each computational programming exercise, this study proposes an approach that enables programming feedback to be provided in real time during online lab session.

MATERIALS AND METHODS

Semantic mistakes: In computational programming exercises, many students may experience with semanticdifficulty. Although, language syntaxes are common errors among students, however, this kind of errors can be rectified by them after having compiler's automated feedback. Feedback is very important to help users to identify their computer statement's errors or mistakes. Meanwhile, semantic errors are more related to illogical meaning of a programming language (e.g., semicolon after condition of a selection, usage of single equal symbol in a comparison) or requirement inaccuracy of a specific question (e.g., inaccurate of looping counter, imprecise of matching condition and mismatch of output's format). These kind of logical errors that depend on specific question's requirement are difficult to be designed into a compiler due to the uniqueness of the question. Without any errors reported based on question requirement, debugging may get harder even after all the general logic's requirement is already satisfied.

Debugging a complete but unaccepted program tends to be difficult especially when the program is free from syntax errors but then failed to be accepted. This is due to unsuccessful output after it was tested with unrevealed test data. If the test data was not revealed to the end users, they may think the automated system is unreliable because their program has successfully acquired a correct output when tested with their own input data. When this happens, they can become frustrated and wondering about the test input. However, in certain case if test input was revealed to the end users, there might be a probability to cheat the system by manipulating the expected output especially when the system was designed based on output matching acceptance.

This study provides a flexible approach to associate feedback on semantic errors of specific computational programming exercise. Expert's workload can be minimized by eliminating solution template and test cases requirement. The feedback can be provided during a lab session or anytime and it will be automatically re-activated as an automated feedback for future similar semantic mistakes.

Program-statement parser: The proposed method does not require solution template to be provided by an expert. However, an expert need to analyze unsuccessful programs that were submitted online. Based on these programs, the expert needs to select certain computer statement that contains mistakes. Then computer statements will be automatically translated into features in the forms of digits, programming symbols (e.g., operators, bracket), keywords (e.g., data types, reserved words) and default library function or method (e.g., println, readInt). Algorithm 1 shows the program-statement extraction rule on the mistake statement to be associated with the expert's feedback. The algorithm will parse the selected statement by reading each character and eliminate unnecessary characters to become a generalized pattern.

Algorithm 1: Program statement extractor (STRING: statement)

```
1   Begin
2     N←Size of selected statement
3     S←List of language symbols
4     K←List of language keywords
5       R←null
6         i←1
7       While (i<N)
8       Begin
9           C = character at location-i
10              If  C←S Then
11                R = R + C
12              Else
13                  BEGIN
14                  Rtemp←null
15                  d←i
16                  While (d<N)
17                  Begin
18                   C = character at location-i
19                   Rtemp = Rtemp+C
20                   Cnext = character at location-(i+1)
21                        If Cnext←S Then
22                        Begin
23                         If Rtemp←(K || digit) Then
24                         R = R+Rtemp
25                         i←d
26                         Break
27                          End
28                           d++
29                           End
30                           End
31                           i++
32                           End
33                           End
34                           Return R
```

For an example of program-statement extraction rule, consider the following Java program:

Java program:

```
1  int y = 5
2  int x = input.next()
3  if (x>y)
4  y = 10
5  if (y = 5)
6  x = 19
```

In line-2, the rule will be extracted as "int =.next();" and associated with an assisted feedback notifying to use "nextInt()" for reading an integer input. Then, in line-3, the rule will be extracted as "if (>)" and associated with assisted feedback notifying that semicolon is not needed at the end of an if statement condition. While in line-5, the statement can be extracted as a feature-statement rule of "if (= 5)" to be associated with assisted feedback alerting that double symbol of "==" needed to be used for comparison instead of single "=". Alternatively, expert user still can modify the automated extracted rule such as by removing the number (e.g., "if (=)") so that the rule will be more general. In order to retrieve the feedback for the specific computer statement's mistake, each line of the computer program needs to be converted into program-statement rule as 1. Then, it will search for any matches of previous program-statement rules. If there are matches, the corresponding of previous assisted feedback will be recommended to the user.

RESULTS AND DISCUSSION

This experiment is meant to evaluate the effectiveness of the feedback in delivering feedback based on program-statement rule of computer program's mistake. It is evaluated based on its capability to extract computer program's statement for a feedback message. This program-statement feedback rule was experimented on the computer programs submission of year 2013 and 2015 covering 475 and 318 of computer programs, respectively. The programs were submitted by first-year computer science student while answering a computational programming question. The question required participant to write a program to read a string input, replaced each of the input characters except the second character into an asterisk and then printed the new string format. By having online submission of a computer program, an expert can choose unsuccessful program and then select any computer statement that contain mistakes. The selected statement will be automatically translated into program-statement rule. Then, the expert user can provide a feedback to be associated with the rule in guiding towards meeting problem's specific requirement. Based on this dataset, feedbacks were provided by an expert user on 8 computer's statements from 7 participants of year 2013.

Table 1: Program-statement feedback rules

Program	Program-statement rule	Feedback

Scanner k = new Scanner (System.in)
Stringay at = k.next()
for (inti = 1;i<ayat. length ();i++){
if(i! = 2){
System.out.print ("*")
}
	for (int = 1;<length ();++){	For loop should start with 0

else{
System.out.print(ayat.charAt (1))
}
}		

Scannerk = new Scanner (System.in)
String messages = k.next()
for (inti = 0;i<messages.length ();I++){
if (i = 1){
System.out.print (messages.char At (1))
}
	if (=1){	Use double == to compare,
		not a single =

else{
System.out.print("*")
}
}		

Scanner k = new Scanner (System.in)
String a = k.next()
for(int i =0; i<length; i++) {
num[i] = k.nextInt ()
message.charAt (0)
System.out.print("p")
else if {
System.out.print ("*")
}
}
	[] = .next Int ();
	Note: Symbol [] was manually removed by the expert to make the rule more general
		This question does not require to read integer the input

for(int i=a.length-1; i>=0; i--) {
System.out.println(num[i])
}		

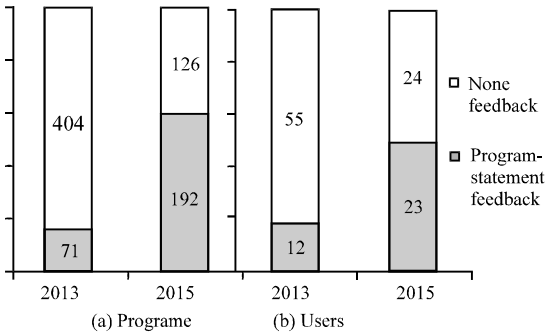

Fig. 1: Feedback coverage on total programs and users

Table 1 shows some examples of provided feedback of computer program's statement. This feedback was tailored to guide end users with the correct usage of the statements towards meeting the specific requirement of a computational programming question. As an example, computer program in the first row, the feedback was given to highlight the correct initialization value for an adequate looping number without changing the existing structure of end user's computer statements. Meanwhile, the second row of computer program, although no syntax errors were reported to the end user, this assisted feedback has managed to highlight the mistake of the assignment operator (=) usage in the condition statement.

Figure 1 shows the statistics on the number of feedback output based on the provided feedback input. Figure 1 shows that, 71 and 192 of submitted computer programs were successfully assisted with the feedback in year 2013 and 2015, respectively. In term of participant's coverage, 12 and 23 participants were assisted with feedback as shown in Fig. 1. As the feedback were provided to 8 users only, others 3 and 22 unattended participants were successfully gain the same feedback of the expert's feedback in year 2013 and 2015, respectively. Thus, the proposed feedback technique has efficiently replicating the expert's feedback input as recommended feedback output to the end users.

Meanwhile, Algorithm 2 shows some examples of automated feedback retrieval based on previous expert's feedback. The full list of the feedback result on computer program submission of year 2013 and 2015 can be downloaded at https://figshare.com/s/942f6ad60982d658a834.

By analyzing the full feedback result of year 2013 and 2015, many users were unaware about their mistake even the same feedback was given repeatedly to the end user on each of their program submission. One of the participants with id-456 has submitted 19 computer programs and 7 out of them were having an input statement mistake. He was using a method to read an integer type input while the question requires a string type input. The user may not alert about the mistake since his submissions on 4 November, 2015 from 10:06:31 am to 10:40:07 am were repeating the same mistake in reading a string input. If this feedback was implemented during the lab session, the student could be notified earlier regarding the mistake and probably could solve the problem more quickly.

Algorithm 2: Feedback based on program-statement rules:

```
Program ID: 456
2015-11-04  10:06:31
0   Import java-util-*
1    Public class App{
2    Public void static main (String [] args)
3    {
4     Scanner yana = new scanner (System-in)
5     String k = yana-nextInt ()
>>RF: This question does not require to read
integer input
6       Char [] = b = k.to Charr Array ()
7       Character  []  characters  =  new
characters [b.length]
8          For (int i = 0; i<b. length; I++)
9          {
10          Characters [i] = b = [i]
11          if (characters [i] ! = characters [1])
12          {
13          System-out-print In ("*")
14           }
15           Else
16           {
17            System-out-print In (characters [1])
18               }
19              }
20          }
21       }
Program ID:  467
2015-11-04  10:40:45
0   Import java-util-Scanner
1   Public class Hangman_Question{
2   Public void static main (String [] args) {
3       Scanner in = new Scanner (System-in)
4
5      String word = in-next ()
6      //char sentence = in next char ()
7
8      For (int i = 0; I<=word-length (); ++) {
>>RF: Use condition of character locaton<
length. Not <= of length since last character is
stored at length-1 location
9                If (sentence-charAt (i) = = charAt (1)}
10                  System-out-print In (charAt (1) )
>>RF: charAt (1) need to be used with string
variable
11               Else
12               System -out-print In ("*")
13
14
15      }
16
17       }
18       }
19
```

In another example, a computer program with id-467 was containing a syntax error of "cannot find symbol of method char At (int)" that provided by a normal Java compiler. This feedback message may not be easily interpreted by a novice user especially when topic on programming method was not yet covered. Alternatively, when specific recommended feedback was implemented during the lab session, this syntax error can be rectified by showing the correct way of using the char At () as a string's method.

Many of such statement mistakes were not part of compiler's roles as it was related to the specific requirement of a question (e.g., inadequate number of loop, mismatch of output formatting and semicolon after selection's condition). As a result, no feedback on the solution's mistake was given to the users as the program could be compiled successfully. Thus, this feedback was significant in helping them to spot the "hidde" mistake and avoid them from being guessing the errors especially when an automated submission system has rejected their solution.

CONCLUSION

Among common approaches to provide automated feedback on computational programming question are dynamic testing, solution template and intelligent agents. However, these approaches not only costly to be built but requires additional workload that need to be prepared from an expert on each computational programming question. This research is using a text-based parser by associating feedback with a filtered computer statement that contain semantic mistakes. It enables an expert to provide instant feedback based on current student's program submission that contain mistake on certain line of codes. Such feedback was later used as an automated feedback based on mistakes of similar selected computer statement's pattern. As computer statements may relate on each other, extending extraction of computer statement to be based on variable accessibility scope is another issue to be taken care in the future. It also will be more effective if expert's feedback can be provided in general in highlighting missing of required instruction logics in a computer program. This feedback can be possibly associated as automated feedback through program similarity technique.

ACKNOWLEDGEMENTS

This research has been supported and funded by Faculty of Information and Communication Technology, Center of Research and Innovation Management (CRIM), Universiti Teknikal Malaysia Melaka and Ministry of Higher Education (MOHE) Malaysia under SLAI scholarship.

REFERENCES

Berdu, L., J. Andre, A. Amandi and M. Campo, 2008. Assisting novice software designers by an expert designer agent. Expert Syst. Appl., 34: 2772-2782.

Denny, P., A.L. Reilly, E. Tempero and J. Hendrickx, 2011. CodeWrite: Supporting student-driven practice of Java. Proceedings of the 42nd ACM Technical Symposium on Computer Science Education, March 09-12, 2011, ACM, Dallas, Texas, ISBN:978-1-4503-0500-6, pp: 471-476.

Gulwani, S., I. Radicek and F. Zuleger, 2014. Feedback generation for performance problems in introductory programming assignments. Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, November 16-21, 2014, ACM, Hong Kong, China, ISBN:978-1-4503-3056-5, pp: 41-51.

Helminen, J., P. Ihantola, V. Karavirta and S. Alaoutinen, 2013. How do students solve parsons programming problems? Execution-based vs. line-based feedback. Proceedings of the Conference on Learning and Teaching in Computing and Engineering (LaTiCE), March 21-24, 2013, IEEE, Macau, China, ISBN:978-1-4673-5627-5, pp: 55-61.

Johnson, W.L., 1990. Understanding and debugging novice programs. Artif. Intell., 42: 51-97.

Kurnia, A., B. Cheang and A. Lim, 2001. Online judge. Comput. Educ., 36: 299-315.

Mungunsukh, H. and Z. Cheng, 2002. An agent based programming language learning support system. Proceedings of the International Conference on Computers in Education, December 3-6, 2002, IEEE, Auckland, New Zealand, ISBN:0-7695-1509-6, pp: 148-152.

Naser, S.A., 2008. An agent based intelligent tutoring system for parameter passing in java programming. J. Theor. Appl. Inf. Technol., 4: 585-589.

Noranis, M.T. and N.S. Azuan, 2013. A multi-agent model for information processing in computational problem solving. Intl. J. Model. Optim., 3: 490-494.

Papancea, A., J. Spacco and D. Hovemeyer, 2013. An open platform for managing short programming exercises. Proceedings of the 9th Annual International ACM Conference on International Computing Education Research, August 12-14, 2013, ACM, San Diego, San California, ISBN:978-1-4503-2243-0, pp: 47-52.

Rafique, U., S.Y. Huang and C.Y. Miao, 2011. Motivation based goal adoption for autonomous intelligent agents. Proceedings of the IEEE-WIC-ACM International Conferences on Web Intelligence and Intelligent Agent Technology Vol. 2, August 22-27, 2011, IEEE, Washington, USA., ISBN:978-0-7695-4513-4, pp: 54-57.

Song, J.S., S.H. Hahn, K.Y. Tak and J.H. Kim, 1997. An intelligent tutoring system for introductory C language course. Comput. Educ., 28: 93-102.

Suarez, M. and R. Sison, 2008. Automatic construction of a bug library for object-oriented novice Java programmer errors. Proceedings of the 9th International Conference on Intelligent Tutoring Systems, June 23-27, 2008, Springer, Montreal, Canada, ISBN:978-3-540-69130-3, pp: 184-193.

Sykes, E.R. and F. Franek, 2003. An intelligent tutoring system prototype for learning to program Java. Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies (ICALT'03), July 9-11, 2003, IEEE, Athens, Greece, ISBN:0-7695-1967-9, pp: 1-5.

Tam, K., 2011. Debugging debugging. Proceedings of the IEEE 35th Annual Conference on Computer Software and Applications Workshop, July 18-22, 2011, IEEE, Munich, Germany, ISBN:978-1-4577-0980-7, pp: 512-515.

Tang, C.M., Y.T. Yu and C.K. Poon, 2010. A review of the strategies for output correctness determination in automated assessment of student programs. Proceedings of the Conference on Global Chinese on Computers in Education (GCCCE), June 01-04, 2010, City University of Hong Kong, Hong Kong, pp: 584-591.

Vaessen, B.E., F.J. Prins and J. Jeuring, 2014. University students' achievement goals and help-seeking strategies in an intelligent tutoring system. Comput. Educ., 72: 196-208.